Chapter 10
함수형 API

함수형 API

함수형 API: 함수 형태로 모델을 구축하는, 모델 구축 방법.

- 층을 생성하고, 각 층의 입력과 출력 관계를 함수처럼 지정하고,
- 마지막에 Model() 에서 전체 모델의 입력층(inputs)과 출력층(outputs)의 인수를 설정해서 모델을 생성한다.
- 모델의 환경설정, 학습, 예측, 평가는 Sequential() 모델과 같이 model.compile(), model.fit(), model.predict(), model.evaluate() 를 사용한다.

함수형 API 모델 예시

```
inputs = Input(shape=(28,28,1))
x = Conv2D(filters=16, kernel_size=(3,3), activation='relu')(inputs)
x = BatchNormalization()(x)
x = MaxPool2D()(x)

x = Conv2D(filters=32, kernel_size=(3,3), activation='relu')(x)
x = MaxPool2D()(x)
x = Dropout(rate=0.2)(x)

x = Flatten()(x)
outputs = tf.keras.layers.Dense(units=10, activation='softmax')(x)
model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

- 각 층을 생성하고, 생성된 층에 대해 input과 output이 함수처럼 지정 된 것을 볼 수 있다.

함수형 API

함수형 API: Permute, Reshape

```
import tensorflow as tf
import numpy as np

#1:
##gpus = tf.config.experimental.list_physical_devices('GPU')
##tf.config.experimental.set_memory_growth(gpus[0], True)

#2: create 2D input data
A = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8 ]],dtype='float32')
A = A.reshape(1, 2, 4, 1)  # (batch, rows, cols, channels)

#3: build a model
x = tf.keras.layers.Input(shape=A.shape[1:])
y = tf.keras.layers.Reshape([4, 2, 1])(x)  # (1, 4, 2, 1)
z = tf.keras.layers.Permute([2, 1, 3])(x)

model = tf.keras.Model(inputs=x, outputs= [y, z])
model.summary()

#4: apply A to model
##output = model(A)    # Tensor output
output = model.predict(A)  # numpy output
print("A[0,:,:,0]=",A[0,:,:,0])
print("output[0]=", output[0][0,:,:,0])  # y
print("output[1]", output[1][0,:,:,0])   # z
```

- Reshape 층은 텐서 형상(shape) 을 변경하고, permute 층은 텐서의 축 순서를 변경한다.
- #2 에서, 2차원 input 데이터 A 를 생성하고, reshape() 을 통해 데이터 shape 을 변경한다.
- #3 에서, 함수형 API를 써서 모델을 생성한다. Y는 reshape 층으로, x의 모양을 (None, 4,2,1) 로 변경한다.
- Z는 permute 층으로, x의 축 순서를 [2,1,3] 으로 변경한다.
  이후, tf.keras.Model() 에서 input과 output을 지정해 준 후 모델을 생성한다. Model은 1개 입력과 2개 출력을 갖는 모델이다.
- #4의 output = model(A) 는 배열 A 를 모델에 투입해서 output을 반환한다.

함수형 API

함수형 API: Permute, Reshape

- 모델 입력 A,
- 모델 출력 y, z

```
A[0,:,:,0]= [[1. 2. 3. 4.]
 [5. 6. 7. 8.]]
output[0]= [[1. 2.]
 [3. 4.]
 [5. 6.]
 [7. 8.]]
output[1] [[1. 5.]
 [2. 6.]
 [3. 7.]
 [4. 8.]]
```

- Output[0].shape 과 output[1].shape 은 모두 (1,4,2,1) 이지만, 텐서에 들어 있는 값은 다르다.

함수형 API

함수형 API: UpSampling1D, UpSampling2D

```python
import tensorflow as tf
import numpy as np

#1:
##gpus = tf.config.experimental.list_physical_devices('GPU')
##tf.config.experimental.set_memory_growth(gpus[0], True)

#2: create 2D input data
A = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8 ]],dtype='float32')
A = A.reshape(1, 2, 4, 1)  # (batch, rows, cols, channels)

#3: build a model
x = tf.keras.layers.Input(shape=A.shape[1:])
y = tf.keras.layers.UpSampling2D()(x) # size = (2,2), [None, 4, 8, 1]

u = tf.keras.layers.Reshape([8, 1])(x)
z = tf.keras.layers.UpSampling1D()(u) # size = 2, [None, 16, 1]

model = tf.keras.Model(inputs=x, outputs= [y, z])
model.summary()

#4: apply A to model
##output = model(A)      # Tensor output
output = model.predict(A) # numpy output
print("A[0,:,:,0]=",A[0,:,:,0])
print("output[0]=", output[0][0,:,:,0])  # y
print("output[1]", output[1][0,:,0])     # z
```

함수형 API

함수형 API: UpSampling1D, UpSampling2D

- 모델 입력 A,
- 모델 출력 y, z

```
1/1 [==============================] - 0s 54ms/step
A[0,:,:,0]= [[1. 2. 3. 4.]
 [5. 6. 7. 8.]]
output[0]= [[1. 1. 2. 2. 3. 3. 4. 4.]
 [1. 1. 2. 2. 3. 3. 4. 4.]
 [5. 5. 6. 6. 7. 7. 8. 8.]
 [5. 5. 6. 6. 7. 7. 8. 8.]]
output[1] [1. 1. 2. 2. 3. 3. 4. 4. 5. 5. 6. 6. 7. 7. 8. 8.]
```

〈u〉

```
array([[[1.],
        [2.],
        [3.],
        [4.],
        [5.],
        [6.],
        [7.],
        [8.]]], dtype=float32)
```

- UpSampling1D 층은 column 방향으로 텐서를 확대한다.
- UpSampling2D 층은 size=(2,2) 일 때, 각각 row 방향으로 텐서 2배 확대, column 방향으로 텐서 2배 확대한다.
- 위 경우, A에 UpSampling2D 층을 적용하면서, row 방향으로 텐서 2배 확대(각 row 벡터 구성하는 스칼라 2배로 복제), Column 방향으로 텐서 2배 확대(세로 방향으로 행렬 구성하는 벡터 2배로 복제) 한다.
- 한편, A를 (8,1) 사이즈로 shape을 변경한 u에 UpSampling1D 층을 적용할 경우, column 방향으로 1차원 벡터(=스칼라)를 2배로 복제한다. 따라서 output[1]의 결과가 나온다.

함수형 API

함수형 API: 1D 병합(merge) 연산

```
1  import tensorflow as tf
2  import numpy as np
3
4  #1:
5  gpus = tf.config.experimental.list_physical_devices('GPU')
6  tf.config.experimental.set_memory_growth(gpus[0], True)
7
8  #2: 1D input data: A, B
9  A = np.array([1, 2, 3, 4, 5]).astype('float32')
10 B = np.array([1, 2, 3, 4, 5, 6, 7, 8]).astype('float32')
11 A = np.reshape(A, (1, -1, 1)) # (batch, steps, channels)
12 B = np.reshape(B, (1, -1, 1)) # (batch, steps, channels)
13
14 #3: build a model
15 input_x = tf.keras.layers.Input(shape=A.shape[1:])
16 input_y = tf.keras.layers.Input(shape=B.shape[1:])
17
18 x = tf.keras.layers.MaxPool1D()(input_x)
19 y = tf.keras.layers.MaxPool1D()(input_y)
20
21 pad = y.shape[1] - x.shape[1] # 2
22 x = tf.keras.layers.ZeroPadding1D(padding=(0, pad))(x)
23
24 out2 = tf.keras.layers.Add()([x, y])
25 ##out2 = tf.keras.layers.Subtract()([x, y])
26 ##out2 = tf.keras.layers.Multiply()([x, y])
27 ##out2 = tf.keras.layers.Minimum()([x, y])
28 ##out2 = tf.keras.layers.Maximum()([x, y])
29 ##out2 = tf.keras.layers.Average()([x, y])
30 out3 = tf.keras.layers.Concatenate()([x, y])
31 out4 = tf.keras.layers.Dot(axes=[1,1])([x, y]) # inner product
32 out5 = tf.keras.layers.Dot(axes=-1)([x, y])    # outer product
33
34 out_list = [x, y, out2, out3, out4, out5]
35 model = tf.keras.Model(inputs=[input_x, input_y], outputs= out_list)
36 ##model.summary()
37 print("model.output_shape=", model.output_shape)
```

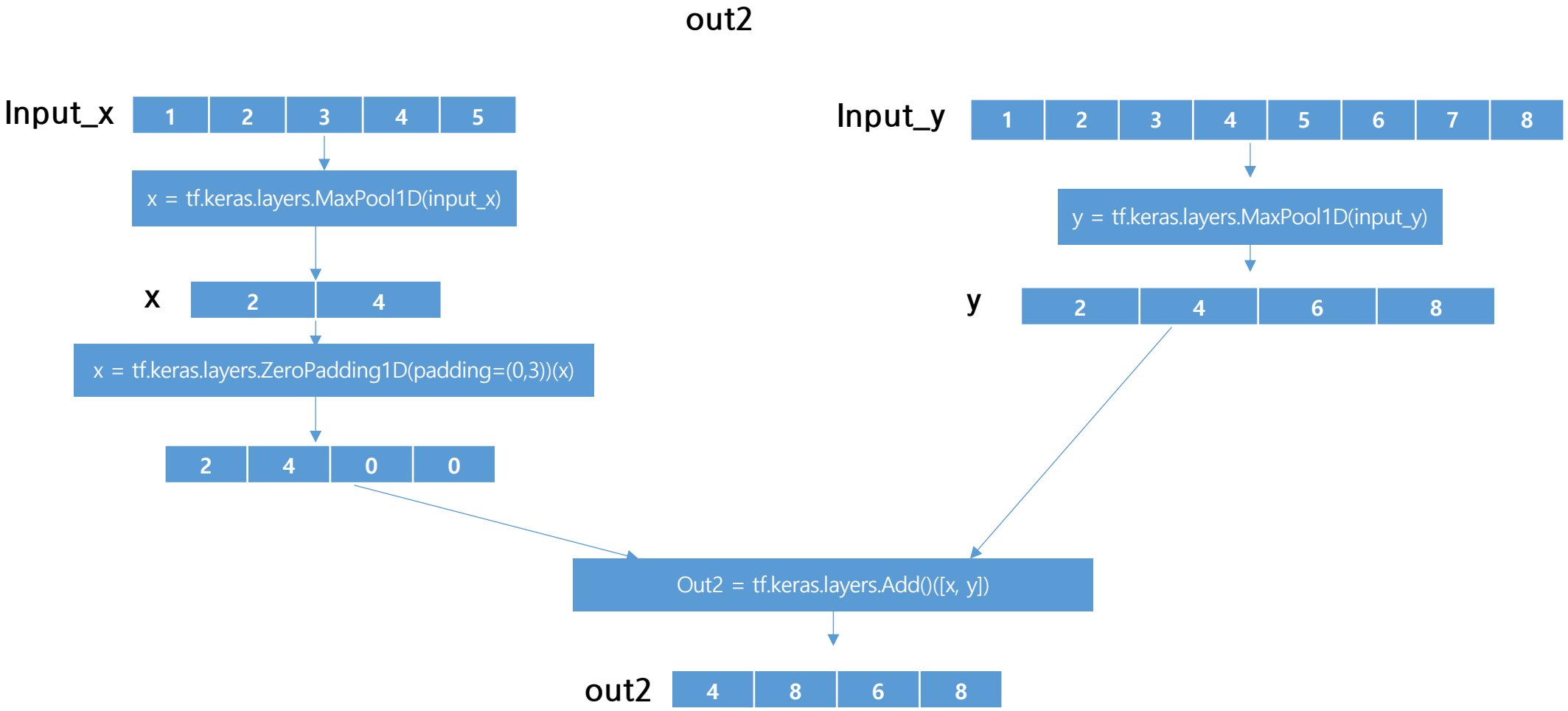
```
39 #4: apply [A, B] to model
40 ##output = model([A, B])      # Tensor output
41 output = model.predict([A, B]) # numpy output
42 for i in range(len(output)):
43     print("output[{}]={}".format(i, output[i]))
44
```

- np.reshape(-1, 정수): reshape 결과에 정수 만큼의 column이 생기도록, 자동으로 구조화 해준다. Input A 의 경우, 정수=1 이므로 A가 1개 column 만 갖도록 A를 변환해준다.
- np.reshape(정수, -1) : reshape 결과에 정수 만큼의 row가 생기도록, 자동으로 구조화 해준다.
- 18번과 19번 줄 코드에서, input_x 와 input_y 에 최대풀링을 적용한 뒤 변수 x, y에 저장한다.
- pad 변수에 패딩 길이 차이를 저장한다. 그리고 x에 ZeroPadding1D 층을 적용해 x 모양을 y와 일치시킨다.
- 24번 줄의 tf.keras.layers.Add() 를 통해, x와 y를 더하고, 그 결과를 out2에 저장한다.
- 30번 줄의 out3은 x, y를 Concatenate() 층을 사용해 '연결'한 결과다.
- 31번 줄의 out4는 x, y의 내적을 계산한 결과이다.
- 32번 줄의 out5는 x, y의 외적을 계산한 결과이다.

함수형 API

함수형 API: 1D 병합(merge) 연산

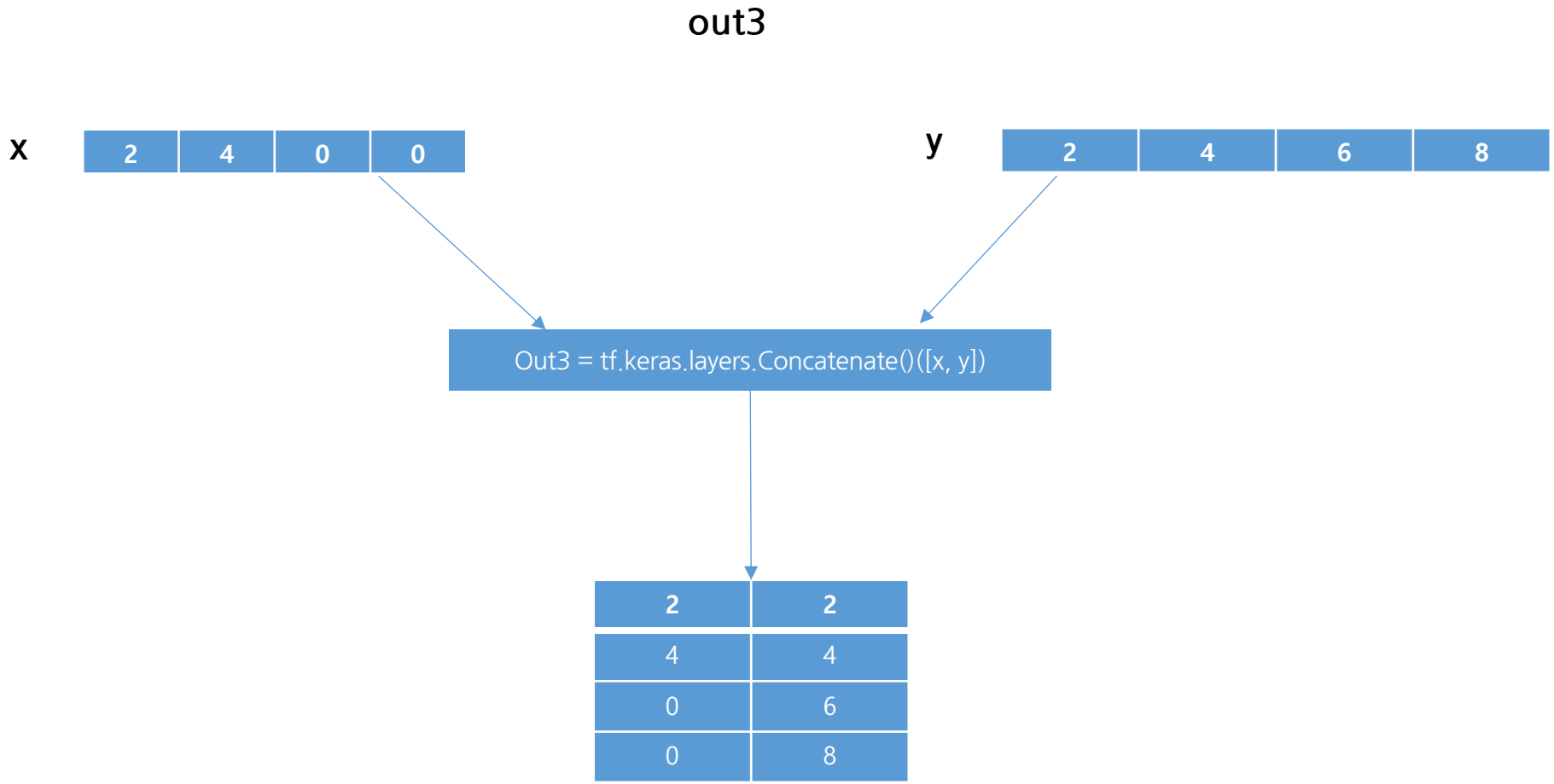
➢ 각 out을 그림으로 표현하면 아래와 같다.



함수형 API

함수형 API: 1D 병합(merge) 연산

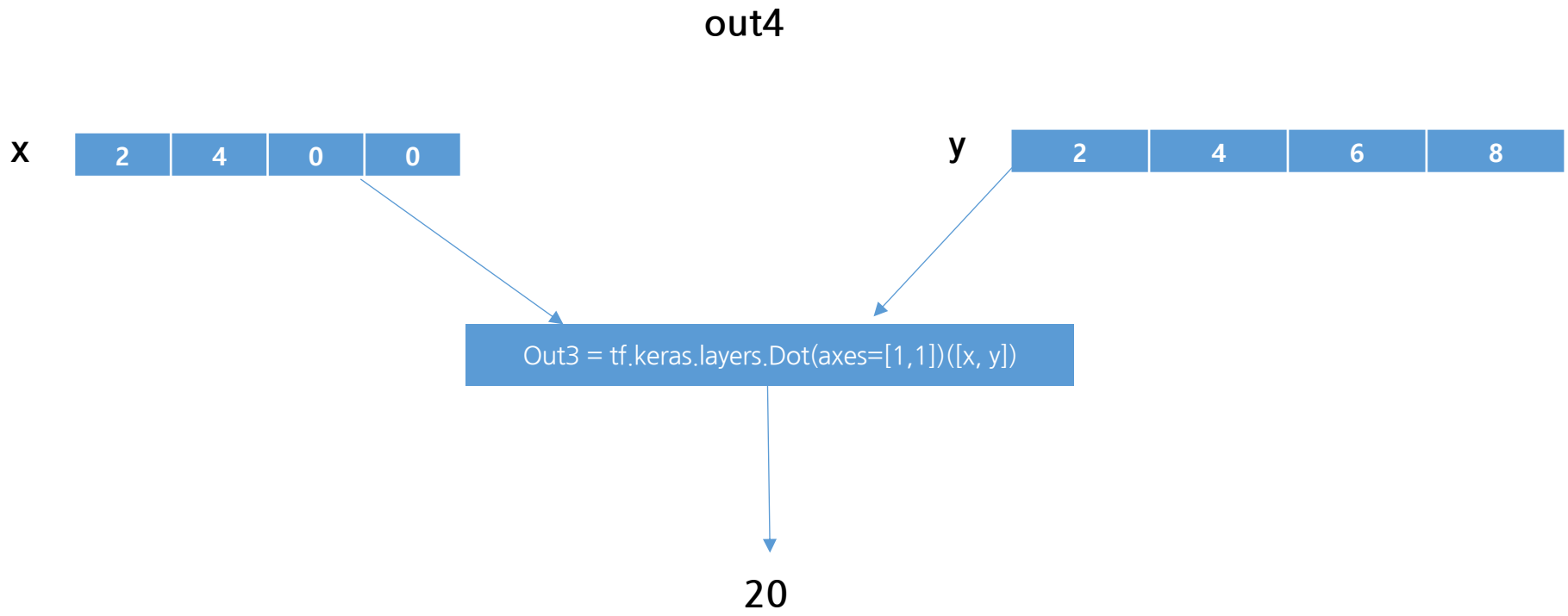
각 out을 그림으로 표현하면 아래와 같다.



함수형 API

함수형 API: 1D 병합(merge) 연산

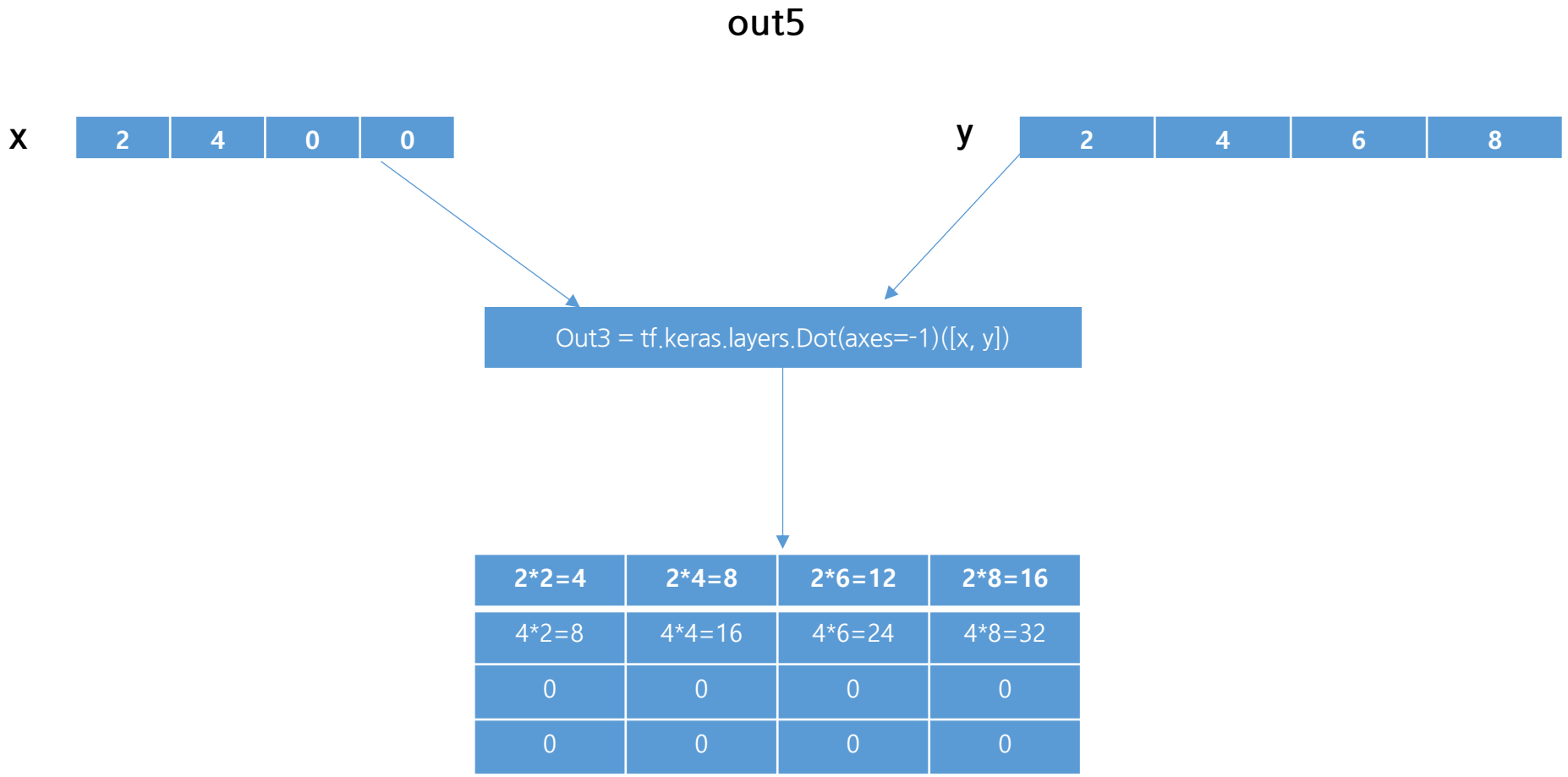
각 out을 그림으로 표현하면 아래와 같다.



함수형 API

함수형 API: 1D 병합(merge) 연산

각 out을 그림으로 표현하면 아래와 같다.



함수형 API

함수형 API: 2D 병합(merge) 연산

```python
import tensorflow as tf
import numpy as np

#1:
#gpus = tf.config.experimental.list_physical_devices('GPU')
#tf.config.experimental.set_memory_growth(gpus[0], True)

#2: 2D input data: A, B, C
A = np.array([[1, 2],
              [3, 4]],dtype='float32')
A = A.reshape(-1, 2, 2, 1) # (batch, rows, cols, channels)

B = np.array([[5, 6],
              [7, 8]],dtype='float32')
B = B.reshape(-1, 2, 2, 1) # (batch, rows, cols, channels)

C = np.array([1, 2, 3]).astype('float32')
C = C.reshape(-1, 3, 1, 1) # (batch, rows, cols, channels)

#3: build a model
x = tf.keras.layers.Input(shape=A.shape[1:]) # shape=(2, 2, 1)
y = tf.keras.layers.Input(shape=B.shape[1:]) # shape=(2, 2, 1)
z = tf.keras.layers.Input(shape=C.shape[1:]) # shape=(3, 1, 1)
out3 = tf.keras.layers.Add()([x, y])
##out3 = tf.keras.layers.Subtract()([x, y])
##out3 = tf.keras.layers.Multiply()([x, y])
##out3 = tf.keras.layers.Minimum()([x, y])
##out3 = tf.keras.layers.Maximum()([x, y])
##out3 = tf.keras.layers.Average()([x, y])
out4 = tf.keras.layers.Concatenate()([x, y])
out5 = tf.keras.layers.Dot(axes= -1)([x, y])   # outer product
out6 = tf.keras.layers.Dot(axes= -1)([x, z])   # outer product
out_list = [x, y, z, out3, out4, out5, out6]
model = tf.keras.Model(inputs=[x, y, z], outputs= out_list)
##model.summary()
print("model.output_shape=", model.output_shape)
```

```python
#4: apply [A, B, C] to model
##output = model([A, B, C])       # Tensor output
output = model.predict([A, B, C]) # numpy output
for i in range(len(output)):
    print("output[{}]={}".format(i, output[i]))
```

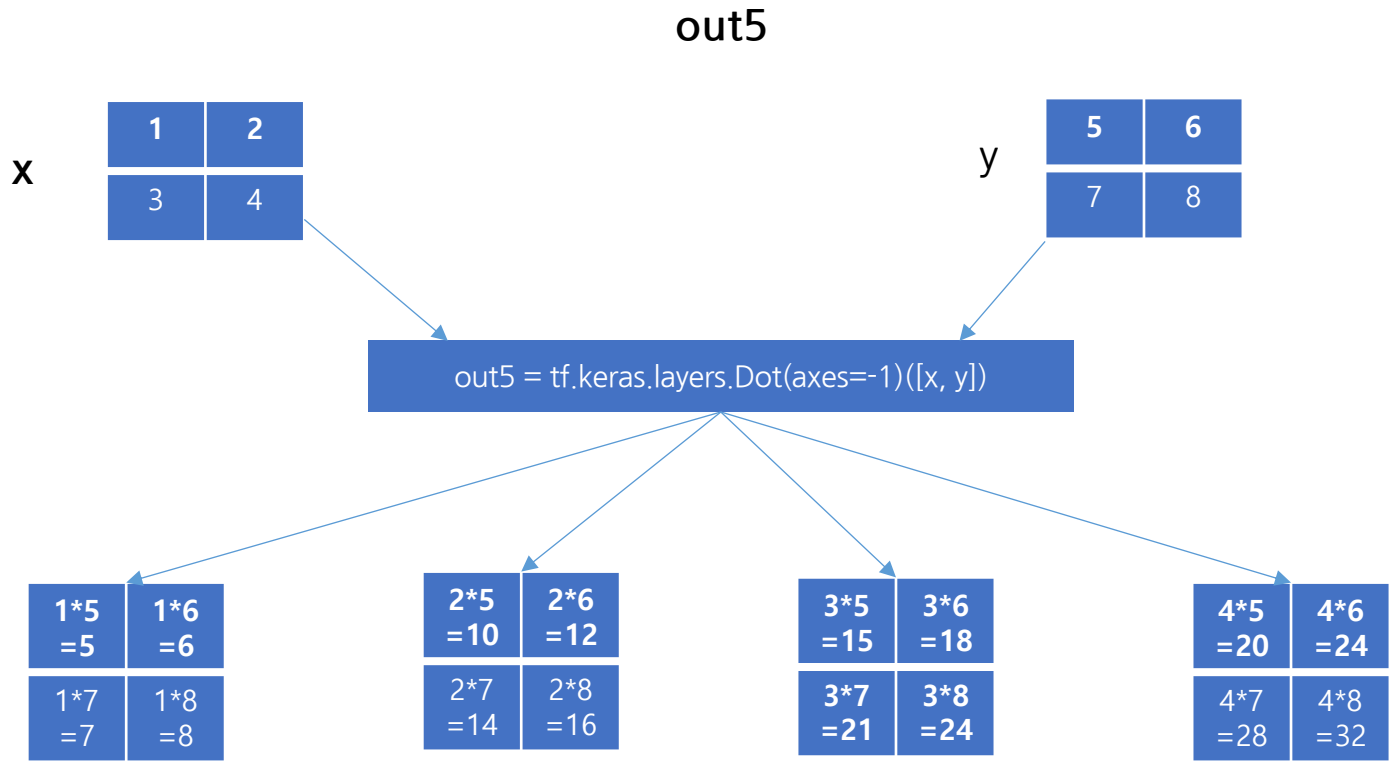
- 모델은 Input 으로 A,B,C 를 입력 받는다.

- out3 은 Add() 층을 통해, x와 y를 합한 값을 담고 있다.
- out4 는 Concatenate() 층을 통해 x와 y를 연결한 값을 담고 있다.
- out5 는 x와 y의 외적을 계산한다.
- out6 는 x와 z의 외적을 계산한다.

- Keras.Model() 은 input으로 x, y, z를 받고, output으로 x, y, z, out3, out4, out5, ou6을 내놓는다.

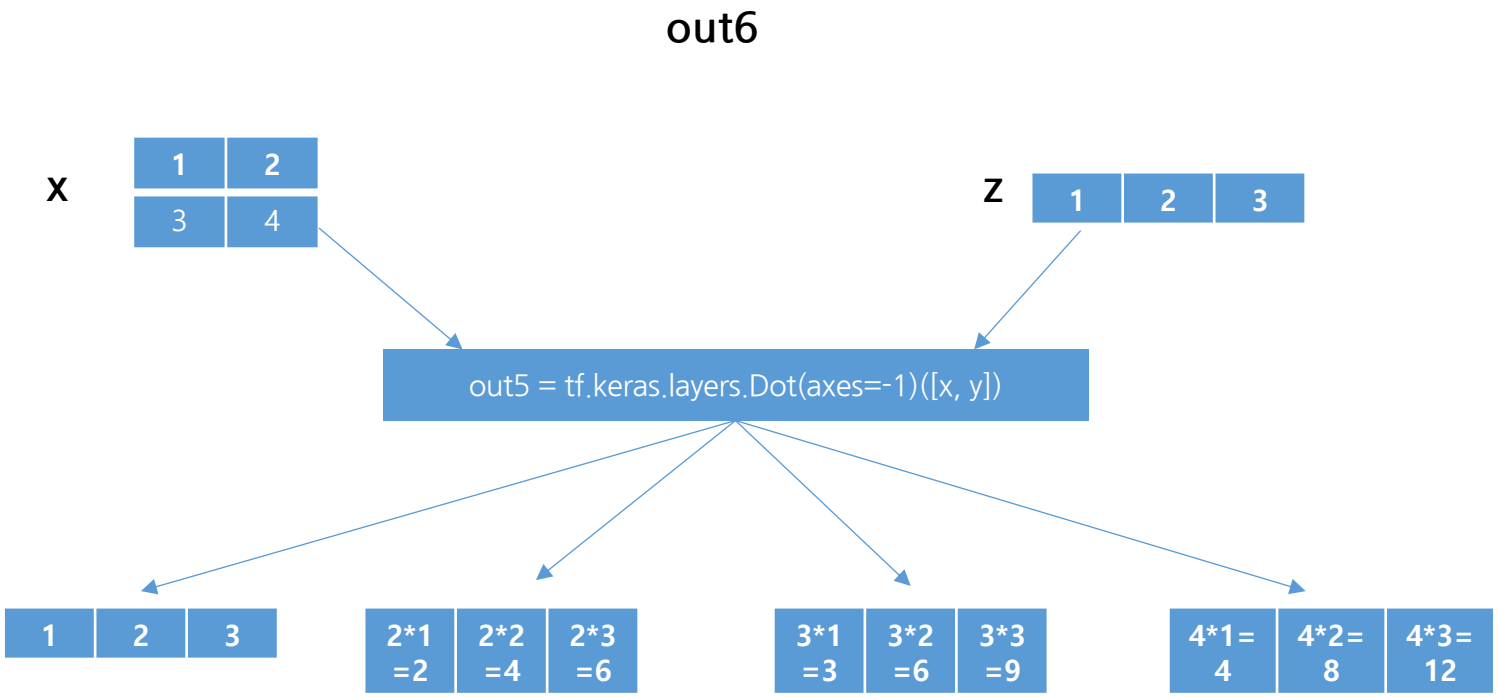
함수형 API

함수형 API: 2D 병합(merge) 연산



함수형 API

함수형 API: 2D 병합(merge) 연산



함수형 API

함수형 API: AND, OR (1 Dense; 1개 퍼셉트론) 문제 해결하기

```python
import tensorflow as tf
from tensorflow.keras.layers     import Input, Dense
import numpy as np
import matplotlib.pyplot as plt

#1: ref [step37_01], [그림 2.9]
##gpus = tf.config.experimental.list_physical_devices('GPU')
##tf.config.experimental.set_memory_growth(gpus[0], True)

#2
X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]], dtype = np.float32)
y_and = np.array([[0],[0], [0],[1]], dtype = np.float32)    # AND
y_or = np.array([[0],[1],[1],[1]], dtype = np.float32)      # OR

#3: build a model
x_and = Input(shape = (2,))
out_and = Dense(units = 1, activation = 'sigmoid', name = 'and')(x_and)

x_or = Input(shape = (2, ))
out_or = Dense(units = 1, activation = 'sigmoid', name = 'or')(x_or)
model = tf.keras.Model(inputs = [x_and, x_or], outputs = [out_and, out_or])
model.summary()
```

층 이름 = 'and'로 표기

층 이름 = 'or'로 표기

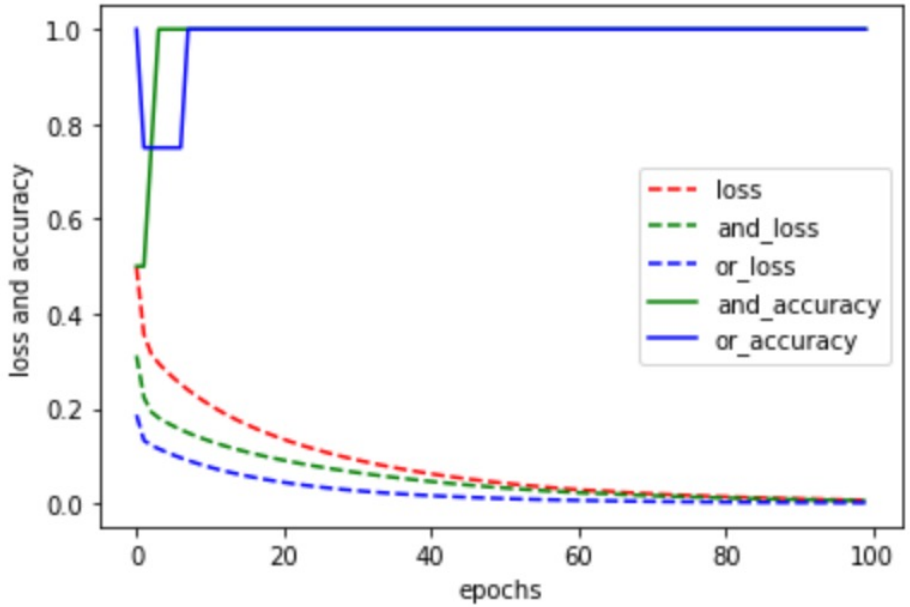
```python
#4: train and evaluate
opt = tf.keras.optimizers.RMSprop(learning_rate = 0.1)
model.compile(optimizer = opt, loss = 'mse', metrics = ['accuracy'])
ret = model.fit(x = [X, X], y = [y_and, y_or],
                epochs = 100, batch_size = 4, verbose = 0)    # silent
test = model.evaluate(x = [X, X], y = [y_and, y_or], verbose = 0)
print('total loss = ', test[0])        # test[1] + test[2]
print('AND: loss={}, acc={}'.format(test[1], test[3]))
print('OR: loss={}, acc={}'.format(test[2], test[4]))

#5: draw graph
plt.plot(ret.history['loss'],     'r--',  label='loss')
plt.plot(ret.history['and_loss'], 'g--', label='and_loss')
plt.plot(ret.history['or_loss'],  'b--', label='or_loss')
plt.plot(ret.history['and_accuracy'], 'g-', label='and_accuracy')
plt.plot(ret.history['or_accuracy'],  'b-', label='or_accuracy')
plt.xlabel('epochs')
plt.ylabel('loss and accuracy')
plt.legend(loc = 'best')
plt.show()
```

- AND 문제) 입력 데이터 X 에서, [1,1] 일 때만 레이블이 1이 붙고, 나머지는 모두 0이다.
- OR 문제) 입력 데이터 X에서, 둘 중 하나만 1 이어도 레이블 1이 붙는다.
- 1개 Dense layer 만 사용해서 AND 문제와 OR 문제를 해결한다.
- Keras.Model()로 다중입력(x_and, x_or)을 받고, 다중출력(out_and, out_or)을 하는 모델을 정의한다.
- 100 epoch 동안 모델을 훈련하며, 미니배치 사이즈는 4를 사용한다.
- 훈련이 끝나면, 훈련 손실과 훈련 정확도(Accuracy rate) 에 대해서 텍스트와 그래프로 결과 출력한다.

함수형 API

함수형 API: AND, OR (1 Dense; 1개 퍼셉트론) 문제 해결하기

```
total loss =  0.006495502777397633
AND: loss=0.005451876670122147, acc=1.0
OR: loss=0.0010436262236908078, acc=1.0
```



- 모델의 훈련결과 나타내는 ret.history는 'loss', 'and_loss', 'or_loss', 'and_accuracy', 그리고 'or_accuracy'를 담고 있다.
- Loss 는 모델의 전체 손실, and_loss 는 and 문제에만 국한해서 계산한 손실, or_loss 는 or 문제에만 국한해서 계산한 손실이다.
- 모델 훈련 결과를 그래프로 시각화 하면 위와 같다.

함수형 API

함수형 API: AND, OR, XOR (2 Dense; 다층 퍼셉트론 사용) 문제 해결하기

```python
import tensorflow as tf
from tensorflow.keras.layers     import Input, Dense
import numpy as np
import matplotlib.pyplot as plt

#1:
##gpus = tf.config.experimental.list_physical_devices('GPU')
##tf.config.experimental.set_memory_growth(gpus[0], True)

#2
X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]], dtype = np.float32)
y_and = np.array([[0],[0], [0],[1]], dtype = np.float32) # AND
y_or  = np.array([[0],[1],[1],[1]], dtype = np.float32)  # OR
y_xor = np.array([[0],[1],[1],[1]], dtype = np.float32)  # XOR

y_and = tf.keras.utils.to_categorical(y_and)
y_or  = tf.keras.utils.to_categorical(y_or)
y_xor = tf.keras.utils.to_categorical(y_xor)

#3: build a model
x_and  = Input(shape=(2,))
x      = Dense(units=2, activation='sigmoid')(x_and)
out_and= Dense(units=2, activation='softmax', name='and')(x)

x_or   = Input(shape=(2,))
x      = Dense(units=2, activation='sigmoid')(x_or)
out_or = Dense(units=2, activation='softmax', name='or')(x)

x_xor  = Input(shape=(2,))
x      = Dense(units=2, activation='sigmoid')(x_xor)
out_xor= Dense(units=2, activation='softmax', name='xor')(x)

model  = tf.keras.Model(inputs = [x_and,   x_or,  x_xor],
                        outputs= [out_and, out_or,  out_xor])
model.summary()
```

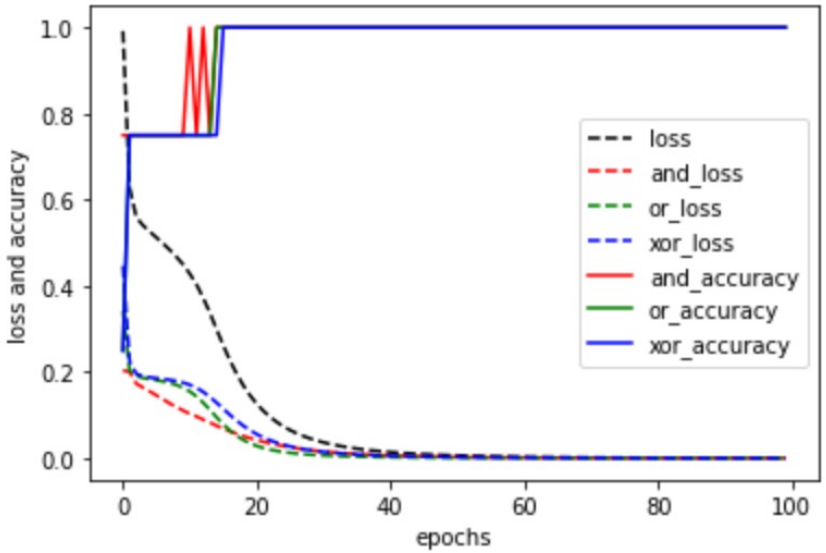
```python
#4: train and evaluate
opt = tf.keras.optimizers.RMSprop(learning_rate=0.1)
model.compile(optimizer=opt, loss='mse', metrics=['accuracy'])
ret = model.fit(x=[X, X, X], y=[y_and, y_or, y_xor],
                epochs=100, batch_size=4, verbose=0)

test = model.evaluate(x=[X, X, X], y=[y_and, y_or, y_xor], verbose=0)
print('total loss = ', test[0]) # test[1] + test[2] + test[3]
print('AND: loss={}, acc={}'.format(test[1], test[4]))
print('OR:  loss={}, acc={}'.format(test[2],  test[5]))
print('XOR: loss={}, acc={}'.format(test[3],  test[6]))

#5: draw graph
plt.plot(ret.history['loss'],    'k--', label='loss')
plt.plot(ret.history['and_loss'],'r--', label='and_loss')
plt.plot(ret.history['or_loss'], 'g--', label='or_loss')
plt.plot(ret.history['xor_loss'],'b--', label='xor_loss')

plt.plot(ret.history['and_accuracy'], 'r-', label='and_accuracy')
plt.plot(ret.history['or_accuracy'],  'g-', label='or_accuracy')
plt.plot(ret.history['xor_accuracy'], 'b-', label='xor_accuracy')
plt.xlabel('epochs')
plt.ylabel('loss and accuracy')
plt.legend(loc='best')
plt.show()
```

- XOR: 두 값이 다르면 1, 두 값이 같으면 0을 리턴 하는 논리회로 이다.
  1개 퍼셉트론으로는 XOR 문제 해결이 불가능했다.
- 하지만, 퍼셉트론 여러 개를 쌓은 다층 퍼셉트론을 사용하면(2개 이상의 완전 연결 층 사용)
  XOR 문제 해결이 가능하다.
- AND 연산은 x_and 로 입력 받고, name='and' 이름의 out_and 로 출력한다.
- OR 연산은 x_or 로 입력 받고, name='or' 이름의 out_or 로 출력한다.
- XOR 연산은 x_xor 로 입력 받고, name='xor' 이름의 out_xor 로 출력한다.
- 세 연산 모두 중간에 히든 유닛 2개를 쓰는 은닉층을 하나씩 갖고 있다. (총 2개 Dense 층 사용)

함수형 API

함수형 API: AND, OR, XOR (2 Dense; 다층 퍼셉트론 사용) 문제 해결하기

```
total loss =  0.0003059952287003398
AND: loss=0.00012597341265063733, acc=1.0
OR:  loss=5.9674344811355695e-05, acc=1.0
XOR: loss=0.00012034746760036796, acc=1.0
```



- 모델 훈련 결과를 각각 모델 전체 loss, and 데이터로만 계산한 and_loss와 and_accuracy, or 데이터로만 계산한 or_loss와 or_accuracy,
- xor 데이터로만 계산한 xor_loss와 xor_accuracy 를 사용해 텍스트와 그래프로 시각화 했다.
- 모델 훈련 결과를 담고 있는 ret.history 딕셔너리는 'loss', 'and_loss', 'or_loss', 'xor_loss', 'and_accuracy', 'or_accuracy', 그리고 'xor_accuracy' 를 담고 있다.

함수형 API 합성곱 신경망

1) IRIS 데이터 분류

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv1D, Dense, Flatten
import numpy as np
import matplotlib.pyplot as plt

#1 : ref [step37_01], 그림 2.9
## pugs = tf.config.experimental.list_physical_devices('GPU')
## tf.config.experimental.set_memory_growth(gpus[0], True)

# 2
def load_Iris(shuffle=False) : # shuffling 하지 않음
    label = {'setosa' : 0, 'versicolor' : 1, 'virginica' : 2} # 붓꽃 각 종류에 대한 레이블 설정
    data = np.loadtxt('./Data/iris.csv', skiprows=1, delimiter = ',', converters = {4:lambda name:label[name.decode()]})

    if shuffle :
        np.random.shuffle(data)
    return data

def train_test_data_set(iris_data, test_rate=0.2) : # train 0.8, test 0.2
    n = int(iris_data.shape[0]*(1-test_rate))
    x_train = iris_data[:n, :-1]
    y_train = iris_data[:n, -1]

    x_test = iris_data[n:,:-1]
    y_test = iris_data[n:,-1]
    return (x_train, y_train), (x_test, y_test)

iris_data = load_Iris(shuffle=True)
(x_train, y_train),(x_test, y_test) = train_test_data_set(iris_data, test_rate=0.2)
## print('x_train.shape:', x_train.shape) # shape=(120,4)
## print('x_test.shape:', x_test.shpe) # shape=(30,4)

# one-hot encoding: 'categorical crossentropy'
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)

# change shapes for conv1d
x_train = np.expand_dims(x_train, axis=2) # shape=(120,4,1)
x_test = np.expand_dims(x_test, axis=2) # shape=(30,4,1)
```

함수형 API 합성곱 신경망

1) IRIS 데이터 분류

```
# 3. build CNN model
def create_cnn1d(input_shape, num_class=3) :
    inputs = Input(shape=input_shape)
    x = Conv1D(filters=10, kernel_size=4, activation='sigmoid')(inputs)
    x = Dense(units=num_class, activation='softmax')(x)

    outputs = Flatten()(x)
    model = tf.keras.Model(inputs=inputs, outputs=outputs)
    return model

model = create_cnn1d(input_shape=(4,1))
model.summary()

# train model
opt = tf.keras.optimizers.RMSprop(learning_rate=0.01)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
ret = model.fit(x_train, y_train, epochs=100, verbose=0)

# evaluate the model
y_pred = model.predict(x_train)
y_label = np.argmax(y_pred, axis=1)
C = tf.math.confusion_matrix(np.argmax(y_train, axis=1), y_label)
print('confusion_matrix(C):', C)

train_loss, train_acc = model.evaluate(x_train, y_train, verbose=2)
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
```

함수형 API 합성곱 신경망

2) MNIST 숫자 분류

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Input, Conv2D, MaxPool2D, Dense
from tensorflow.keras.layers import BatchNormalization, Dropout, Flatten
from tensorflow.keeras.optimizers import RMSprop
import numpy as np
import matplotlib.pyplot as plt

# 1 ref: step37_01, 그림 2.9
gpus = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(gpus[0], True)

# 2
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255.0
x_test /=255.0

# expand data with channel=1
x_train = np.expand_dims(x_train, axis=3) # (60000,28,28,1)
x_test = np.expand_dims(x_test, axis=3) # (10000,28,28,1)

# one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)

# 3 build cnn
def create_cnn2d(input_shape, num_class = 10) :
    inputs=Input(shape=input_shape)
    x = Conv2D(filters=16, kernel_size=(3,3), activation='relu')(inputs)
    x = BatchNormalization()(x)
    x = MaxPool2D()(x)
    x = Conv2D(filters=32, kernel_size=(3,3), activation='relu')(x)
    x = MaxPool2D(x)
    x = Dropout(rate=0.2)(x)

    x = Flatten()(x)

    outputs=tf.keras.layers.Dense(units=10, activation='softmax')(x)
    model=tf.keras.Model(inputs=inputs, outputs=outputs)
    return model

model = create_cnn2d(input_shape=x_train.shape[1:])
```

함수형 API 합성곱 신경망

2) MNIST 숫자 분류

```
# train the model
# train model
opt = tf.keras.optimizers.RMSprop(learning_rate=0.01)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
ret = model.fit(x_train, y_train, epochs=100, batch_size=400, verbose=0)

# evaluate the model
y_pred = model.predict(x_train)
y_label = np.argmax(y_pred, axis=1)
C = tf.math.confusion_matrix(np.argmax(y_train, axis=1), y_label)
print('confusion_matrix:', C)

train_loss, train_acc = model.evaluate(x_train, y_train, verbose=2)
test_loss, test_acc = model.evalutate(x_test, y_test, verbose=2)
```

함수형 API 합성곱 신경망

3) CIFAR-10 분류

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.layers import Input, Conv2D, MaxPool2D, Dense
from tensorflow.keras.layers import BatchNormalization, Dropout, Flatten
from tensorflow.keras.optimizers import RMSprop
import numpy as np
import matplotlib.pyplot as plt

# 1. ref step 37_01 그림 2.9
gpus = tf.config.experimental.list_physical_devices('GPU')



# 2.
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype('float32') # (50000, 32, 32, 3)
x_test = x_test.astype('float32') # (10000, 32, 32, 3)

# one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)

# 3
def normalize_image(image) : # 3-channel image
    mean = np.mean(image, axis=(0,1,2))
    std = np.std(image, axis=(0,1,2))
    image = (image-mean)/std
    return image
x_train = normalize_image(x_train) # range: N(mean, std)
x_test = normalize_image(x_test)

# 4 build model
def create_cnn2d(input_shape, num_class=10) :
    inputs = Input(shape=input_shape) # shape = (32, 32, 3)
    x = Conv2D(filters=16, kernel_size=(3,3), activation='relu')(inputs)
    x = BatchNormalization()(x)
    x = MaxPool2D()(x)

    x = Conv2D(filters=32, kernel_size=(3,3), activation='relu')(x)
    x = MaxPool2D()(x)
    x = Dropout(rate=0.2)(x)

    x = Flatten()(x)
    outputs = Dense(units=num_class, activation='softmax')(x)
    model = tf.keras.Model(inputs, outputs)
    return model

model = create_cnn2d(input_shape=x_train.shape[1:])
```

함수형 API 합성곱 신경망

3) CIFAR-10 분류

```
# train the model
opt = RMSprop(learning_rate=0.01)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
ret = model.fit(x_train, y_train, epochs=100, batch_size=10, validation_data=(x_test, y_test), verbose=0)

# evaluate the model
y_pred = model.predict(x_train)
y_label = np.argmax(y_pred, axis=1)
C = tf.math.confusion_matrix(np.argmax(y_train, axis=1), y_label)
print('confusion matrix:', C)
train_loss, train_acc = model.evaluate(x_train, y_train, verbose=2)
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)

# plot
fig, ax = plt.subplots(1,2, figsize=(10,6))
ax[0].plot(ret.history['loss'], 'g-')
ax[0].set_title('train loss')
ax[0].set_xlabel('epochs')
ax[0].set_ylabel('loss')

ax[1].plot(ret.history['accuracy'], 'b-', label='train accuracy')
ax[1].plot(ret.history['val_accuracy'], 'r', label='test accuracy')
ax[1].set_title('accuracy')
ax[1].set_xlabel('epochs')
ax[1].set_ylabel('accuracy')
plt.legend(loc='best')
fig.tight_layout()
plt.show()
```

함수형 API 합성곱 신경망

4) CIFAR-100 분류

```
## step44_04
import tensorflow as tf
from tensorflow.keras.datasets import cifar100
from tensorflow.keras.layers import Input, Conv2D, MaxPool2D, Dense
from tensorflow.keras.layers import BatchNormalization, Dropout, Flatten
from tensorflow.keras.optimizers import RMSprop
import numpy as np
import matplotlib.pyplot as plt

# 1. ref
gpus = tf.config.experimental.list_physical_devices('GPU')

# 2
(x_train, y_train), (x_test, y_test) = cifar100.load_data()
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)

# 3
def normalize_image(image) : # 3-channel image
    mean = np.mean(image, axis=(0,1,2))
    std = np.std(image, axis=(0,1,2))
    image = (image-mean)/std
    return image
x_train = normalize_image(x_train) # range: N(mean, std)
x_test = normalize_image(x_test)

# 4. build model
def create_cnn2d(input_shape, num_class=100) :
    inputs = Input(shape=input_shape) # shape : (32,32,3)
    x = Conv2D(filters=16, kernel_size=(3,3), padding='same', activation='relu')(inputs)
    x = BatchNormalization()(x)
    x = MaxPool2D()(x)

    x = Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu')(x)
    x = BatchNormalization()(x)
    x = MaxPool2D()(x)
    x = Dropout(rate=0.25)(x)

    x = Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu')(x)
    x = BatchNormalization()(x)
    x = MaxPool2D()(x)
    x = Dropout(rate=0.5)(x)

    x = Flatten()(x)
    outputs = Dense(units=num_class, activation='softmax')(x)
    model = tf.keras.Model(inputs, outputs)
    return model
model = create_cnn2d(input_shape=x_train.shape[1:])
model.summary()
```

함수형 API 합성곱 신경망

4) CIFAR-100 분류

```
# train model
opt = RMSprop(learning_rate=0.001)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
ret = model.fit(x_train, y_train, epochs=200, batch_size=400, validation_data=(x_test, y_test), verbose=0)

# evaluate model
y_pred = model.predict(x_train)
y_label = np.argmax(y_pred, axis=1)
C = tf.math.confusion_matrix(np.argmax(y_train, axis=1), y_label)
print('confusion matrix:', C)
train_loss, train_acc = model.evaluate(x_train, y_train, verbose=2)
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)

# plot
fig, ax = plt.subplots(1,2, figsize=(10,6))
ax[0].plot(ret.history['loss'], 'g-')
ax[0].set_title('train loss')
ax[0].set_xlabel('epochs')
ax[0].set_ylabel('loss')

ax[1].plot(ret.history['accuracy'], 'b-', label='train accuracy')
ax[1].plot(ret.history['val_accuracy'], 'r-', label='test accuracy')
ax[1].set_title('accuracy')
ax[1].set_xlabel('epochs')
ax[1].set_ylabel('accuracy')
plt.legend(loc='best')
fig.tight_layout()
plt.show()
```